Vue中的组件化开发

**# 如何创建组件？**

  在src中，我们创建一个 Xxx.vue 的文件，就是创建一个vue的组件！

  在此组件中包含：构建的视图「基于<template>语法」、CSS样式、JS程序

  我们把这样的组件称之为“单文件组件”！

  ----

  例如：App.vue

  // 构建的视图：要求视图中只能有一个“根节点”「Vue2是这样的，Vue3可以出现多个根节点」

  <template>

    <div id="app">

        珠峰培训

    </div>

  </template>

**# JS程序：按照Vue2中的OptionsAPI，开发相关的程序**

  <script>

    // 导出的对象，和之前new Vue传递进来的options配置项是一样的

    export default {

       // 组件名

       name: "App",

       // 组件中需要使用的响应式数据(状态)

       // 为啥要写成函数(闭包)？目的是保证组件合并渲染的时候，各个组件之间的状态不会冲突

       data(){

          return {

            ...

          };

       },

       // 普通方法

       methods:{},

       // 计算属性

       computed:{},

       // 监听器

       watch:{},

       // 过滤器

       filters:{},

       ...

    };

  </script>

  // 构建组件需要的样式

  // + lang:language 设置CSS预处理器语言，不设置只能按照CSS方式编写

  // + scoped:设置样式私有化；一但设置这个属性，此处写的样式只对当前组件生效，如果不设置，则是全局的样式（这样可能导致组件合并的时候，组件之间的样式冲突）！

  <style lang="less" scoped>

    ...

  </style>

  ---

  为了让vscode支持单文件组件的语法（能够有提示和进行代码格式化对齐），我们需要安装指定的插件：

  + vetur

  + Vue Language Features (Volar)

**# Vue中的组件化开发**

如何创建组件？

  在src中，我们创建一个 Xxx.vue 的文件，就是创建一个vue的组件！

  在此组件中包含：构建的视图「基于<template>语法」、CSS样式、JS程序

  我们把这样的组件称之为“单文件组件”！

  ----

  例如：App.vue

  // 构建的视图：要求视图中只能有一个“根节点”「Vue2是这样的，Vue3可以出现多个根节点」

  <template>

    <div id="app">

        珠峰培训

    </div>

  </template>

**# JS程序：按照Vue2中的OptionsAPI，开发相关的程序**

  <script>

    // 导出的对象，和之前new Vue传递进来的options配置项是一样的

    export default {

       // 组件名

       name: "App",

       // 组件中需要使用的响应式数据(状态)

       // 为啥要写成函数(闭包)？目的是保证组件合并渲染的时候，各个组件之间的状态不会冲突

       data(){

          return {

            ...

          };

       },

       // 普通方法

       methods:{},

       // 计算属性

       computed:{},

       // 监听器

       watch:{},

       // 过滤器

       filters:{},

       ...

    };

  </script>

  // 构建组件需要的样式

  // + lang:language 设置CSS预处理器语言，不设置只能按照CSS方式编写

  // + scoped:设置样式私有化；一但设置这个属性，此处写的样式只对当前组件生效，如果不设置，则是全局的样式（这样可能导致组件合并的时候，组件之间的样式冲突）！

  <style lang="less" scoped>

    ...

  </style>

  ---

  为了让vscode支持单文件组件的语法（能够有提示和进行代码格式化对齐），我们需要安装指定的插件：

  + vetur

  + Vue Language Features (Volar)

组件命名规范

  + 大驼峰命名法 CameCase/PascalCase

  + kebab-case

Vue组件开发的第一要素，就是组建的划分

+ 普通业务逻辑

  把复杂的页面进行拆分，拆分成一个个的组件，不会服用的组件就是普通业务组件在SPA单页面应用中，每一个页面都会被开发为一个组件，这也是普通业务组件

+ 通用业务组件

  会被复用的组件

+ 功能型组件[一般都是具备很强的复用性]

  如果是自己开发的，也放在components中

  一般我们都是直接使用UI组件库中的组件即可

    PC:element-ui/plus、antd for vue、iview...

    移动端:vant、cube

**# Vue组件的划分**

  + 局部组件

    需要调用这个组件，首先要导入

  + 全局组件

    我们可以基于Vue.component这个方法，注册组件为全局组件，这样在任何其他组件中，像调用直接调用即可，无需再导入注册

  + 普通业务组件一般都是局部组件：通用业务组件/功能组件，一般设置为全局组件（通用业务组件设置为局部组件的也很多，但是通用功能组件[UI组件库中的]一般都是全局组件）

再Vue2中，正常情况下，每创建一个组件[XxxXxx.vue]，都是创建一个类[VueComponent]，而且这个类是Vue的子类

  每调用一次组件，都是创建这个类的一个实例

    + 实例具备很多私有属性

    + 实例.*\_proto\_* -> VueComponent.prototype -> Vue.prototype上的方法

  -----从从调用组件开始，到组件渲染完毕，再到组件内部状态更改控制试图更新，最后到组件销毁释放，有一套完整的处理机制和流程

  @1 创建组件类(或Vue)的一个实例

  @2 触发周期函数(钩子函数)：beforeCreate

  @3 初始化实例：项实例上挂载需要的东西

    + 接受/注册传递进来的属性值（函数型规则校验），冰球把获取的属性值做数据劫持，最后挂载在实例上

    + 接受/注册上下文信息

    + 对data中的数据进行"深度劫持"，最后挂载在实例上

    + 处理计算属性、监听器、methods等内容

    + 设置组件名字、注册组件...

    + ...

  @4 触发钩子函数：created

**#**

  @5 触发钩子函数：beforeMount 第一次渲染之前

    + 一般我们再这两个钩子函数中一个里[一般都是created]，回想服务器发送异步的数据请求，这样可以保证那个第一次渲染完成后，可以尽快获取到真实的数据，再让组件更新即可

  @6 组件开始进行渲染

    + 首先基于vue-template-compiler，把<template>视图编译为虚拟DOM对象

    + 再把虚拟DOM渲染为真实的DOM

  @7 触发钩子函数：mounted 第一次渲染完毕

    我们一般会在第一次渲染完毕后：

      + 获取DOM元素，直接去操作DOM[例如：修改内容或样式、获取相关信息、事件绑定等]

      + 设置定时器做一些处理 3

**# 当组件的状态被更改，组件开始更新的时候：**

  @1 触发钩子函数：beforeUpdate 组件更新之前

  @2 组件开始更新

  \*

    + 按照最新的数据，把视图编译为虚拟DOM对象[全部重新生成]

    + DOM-DIFF：会用最新的vnode和之前的vnode做对比，算出差异的部分[补丁包]

    + 最后把补丁包渲染为真是DOM，一次实现更新即可

  @3 触发钩子函数：updated 组件更新之后

  @4 触发this.$nextTick设置的回调函数

    \*

    我们偶尔也会在组件更新完毕后，做一些事情

      + updated：不论修改的那个状态，都会触发

      + $nextTick：可以指定，哪个状态更改后触发

**# 当组件被释放销毁的时候**

  @1 触发钩子函数：beforeDestory

  \*

    我们一般会在销毁之前，把一些默认不会移除的内容，手动移除，例如：

      + 自己设置的定时器或者监听器

      + 自己基于addEventListener设置的事件绑定

      + ...

    还可以把一些信息保存起来[例如：表单信息草稿箱]你

  @2 组件开始销毁

    + 生成的虚拟DOM（vnode）会被释放掉

    + 基于v-on（或@）绑定的事件会被移除掉

    + 页面中的真实DOM会被移除掉

  @3 触发钩子函数：destoryed

**# 基于ref获取DOM元素[非受控组件]**

  视图

  <p ref="ratioBox">...</p>

  程序：mounted/updated

  this.$refs 获取的嘘一个对象，对象中存储获取的DOM元素

  {

    ratioBox：获取DOM元素

  }

  this.$refs.ratioBox

  ---

  受控组件：由状态驱动视图变化 “受状态管控的组件”

  非受控组：获取DOM元素直接进行操作的

  ---

  ref除了获取DOM元素之外，还可以获取组件的实例

    + 把ref赋值给调用的组件

    + this.$ref.xxx 获取的是当前调用组件所创建的实例

**# this.$nextTick**

  它是Vue.prototype上的方法

  this.$nextTick(callback)：在状态更改，视图更改后，触发回调函数执行，晚触发于updated

  ----

  每一次状态更改，都是“异步操作”：修改完状态后，并没有立即执行“更新逻辑”，而是等到当前上下文中，同步事情处理完毕，再去开启更新的流程

    + 这样设计的好处：同时更改多个状态值，也不会一次次的触发更新，二十同意“批处理”，只更新一次！大大提高了性能

**# scoped可以保证单文件组件中，样式的私有化，防止组件合并后，样式冲突问题**

  实现原理：

  \* 每个组件都会有一个"唯一的ID"，例如：<div data-v-7ba5bd90="" class="demo-box">

    + 同一个组件调用N次，ID也是相同的

    + 创建组件就存在则个ID

  \* 一但我们给<style>添加了scoped

    + 在当前组件<template>中编写的元素都会默认加一个自定义属性[属性的ID是唯一的]

      <div data-v-78d2b2e9 class="demo-box">  ... <div>

      PS：只有写在<template>中的元素才会加这个属性

    + 并且在<style>中编写的所有样式，都会默认设置一个属性选择器[选择器内容是组件的ID]

      .demo-box[data-v-78d2b2e9]{

        ...

      }

  这就是组件样式私有化的原理

  -----

  但是样式私有化之后，也会遗留一个“大坑”：

    + 只有在当前组件<template>中写的元素，才会加这个ID属性

    + 调用一个其他组件，调用组件的最外层元素也会加这个ID属性，但是组件内部的元素，因为并不是在此组件的<template>中编写的，所以“不会给这些元素设置ID属性”

    + 但是如果在此组件的<style>中，想要调用组件的内部元素样式，就会出现问题：

      首先我们写的样式，因为在<style scoped>中，所以都会加一个属性选择器

      但是调用组件的内部元素，并没有设置这个ID属性

      导致设置的样式无法生效

  我们此时需要给编写的样式，设置“样式穿透”[/deep/]：这样会去掉我们编写样式的ID属性选择器HTML没有设置ID属性，我们写的样式也没有属性选择器，这样样式就可以生效了

**# 关于封装组件“强复用性”设计的知识点**

  \* 封装组件的目的：让其具备更强的复用性，后期调用组件的时候，通过传递不同的信息，让组件呈现出我们想要的效果

  props属性

    \* 调用组建的时候，可以传递不同的属性值

        + 但凡传递的属性值不是字符串类型[比如：布尔、数字、把状态值作为属性值...]，都需要基于v-bin(简写:)

        + 如果传递的属性值是style/class这样的内置属性，会自动给组件最外层元素设置对应的属性

        + 如果只写了属性名，但是没有设置属性值，则默认传递的属性值是空字符串“ ”

    \* 组件内部可以接收这些属性值，并且进行规则校验；根据传递不同的属性值，让组件呈现出不同好的效果

      \* props: ["size", "type"]

        + 不管传递的属性有多少，我们只接受两个；并且把这两个属性，劫持和挂在到实例上；

        + 如果我们接收的属性没有传递，也会劫持和挂载，只不过属性值是undefined

        + 不能接收注册class/style [也无需接收]

        + this.$attrs：获取到传递的属性中，没有被props注册的，而且排除class/style，其余的属性可在这获取

        + this.$listeners：获取调用组件时基于v-on(或@)绑定的自定义事件

  @2 slot插槽

  \*

    封装组件的时候，我们先预留出相关的位置，由调用组件时候，用户传递给我们

      + 如何使用插槽

      + 具名插槽

      + 作用域插槽

  @3 自定义属性 $on / $off / $emit

  \*

    + 调用组建的时候，可以基于v-on，给组件实力的自定义事件池中，绑定一些自定义事件和方法；再组件内部，基于某些操作，通知这些自定义事件方法执行，从而对应

module.exports = {

  presets: [

    ['@vue/cli-plugin-babel/preset', {

      useBuiltIns: 'usage'

    }]

  ]

}

{

  "compilerOptions": {

    "target": "es5",

    "module": "esnext",

    "baseUrl": "./",

    "moduleResolution": "node",

    "paths": {

      "@/\*": [

        "src/\*"

      ]

    },

    "lib": [

      "esnext",

      "dom",

      "dom.iterable",

      "scripthost"

    ]

  }

}

{

  "name": "vuedemo",

  "version": "0.1.0",

  "private": true,

  "scripts": {

    "serve": "vue-cli-service serve",

    "build": "vue-cli-service build",

    "lint": "vue-cli-service lint"

  },

  "dependencies": {

    "axios": "^0.27.2",

    "blueimp-md5": "^2.19.0",

    "core-js": "^3.8.3",

    "echarts": "^5.3.3",

    "element-ui": "^2.15.9",

    "qs": "^6.11.0",

    "vue": "^2.6.14",

    "vue-router": "^3.6.4",

    "vuex": "^3.6.2"

  },

  "devDependencies": {

    "@babel/core": "^7.12.16",

    "@babel/eslint-parser": "^7.12.16",

    "@vue/cli-plugin-babel": "~5.0.0",

    "@vue/cli-plugin-eslint": "~5.0.0",

    "@vue/cli-service": "~5.0.0",

    "babel-plugin-component": "^1.1.1",

    "eslint": "^7.32.0",

    "eslint-plugin-vue": "^8.0.3",

    "less": "^4.1.3",

    "less-loader": "^8.1.1",

    "vue-template-compiler": "^2.6.14"

  },

  "eslintConfig": {

    "root": true,

    "env": {

      "node": true

    },

    "extends": [

      "plugin:vue/essential",

      "eslint:recommended"

    ],

    "parserOptions": {

      "parser": "@babel/eslint-parser"

    },

    "rules": {}

  },

  "browserslist": [

    "> 1%",

    "last 2 versions",

    "not ie<=9"

  ]

}

const { defineConfig } = require('@vue/cli-service')

const env = process.env.NODE\_ENV

module.exports = defineConfig({

  publicPath: './',

  lintOnSave: env === 'development' ? true : false,

  transpileDependencies: false,

  productionSourceMap: false,

  devServer: {

    host: '127.0.0.1',

    open: true,

    proxy: {

      '/api': {

        target: 'https://www.jianshu.com/asimov',

        changeOrigin: true,

        ws: true,

        pathRewrite: { '^/api': '' }

      }

    }

  }

})

<template>

  <div id="app">

    组件内部

    <DemoOne />

    <demo-Two />

    <!-- <slot-demo>

      哈哈哈

      <span>===</span>

      <span>+++</span>

      <template #header>

        我是头部的信息

        <span>---</span>

      </template>

      <template #footer>

        我是头部的信息

        <span>---</span>

      </template>

    </slot-demo> -->

    <slot-demo>

      <!--

            v-slot="插槽名=‘变量’

            dufault可以忽略

            AA是变量

            #dufault=‘变量’ 名字无法忽略

            v-slot="{x,y}" 直接解构赋值

        " -->

      <template v-slot="AA">

        <span>我是插槽信息</span>

        {{ AA.x + AA.y }}

      </template>

    </slot-demo>

  </div>

</template>

<script>

import DemoOne from "@/views/DemoOne.vue";

import DemoTwo from "@/components/DemoTwo.vue";

import SlotDemo from "./views/SlotDemo.vue";

export default {

  components: { DemoOne, DemoTwo, SlotDemo },

  name: "App",

};

</script>

<style lang="less">

</style>

、<template>

  <div class="demo-box">

    <p>支持：{{ supNum }}</p>

    <p>反对：{{ oppNum }}</p>

    <p ref="ratioBox">支持率：{{ ratio }}</p>

    <div>

      <button @click="change('sup')">支持</button>

      <button @click="change('opp')">反对</button>

    </div>

  </div>

</template>

<script>

export default {

  name: "DemoOne",

  data() {

    return {

      supNum: 10,

      oppNum: 5,

    };

  },

  methods: {

    change(type) {

      if (type === "sup") {

        this.supNum++;

        this.$nextTick(() => {

          console.log("支持数累加");

        });

        return;

      }

      this.oppNum++;

    },

  },

  computed: {

    ratio() {

      let { supNum, oppNum } = this,

        total = supNum + oppNum;

      return total === 0 ? "--" : ((supNum / total) \* 100).toFixed(2) + "%";

    },

  },

  mounted() {},

  updated() {

    console.log("试图更新完毕");

  },

};

</script>

<style lang="less">

.demo-box {

  color: red;

}

</style>

<template>

  <div class="slot-demo">

    组件内部{{ x + y }}

    <br />

    <slot :x="x" :y="y"></slot>

  </div>

</template>

<script>

export default {

  name: "slot-demo",

  data() {

    return {

      x: 10,

      y: 20,

    };

  },

  mounted() {},

  methods: {},

};

</script>

<style lang="less" scoped>

</style>

<template>

  <div class="demo-box">

    <h2 class="title" @click="change('aahahah')">DEMO2组件</h2>

    <!--

        size:控制按钮大小 'large'、'small（默认值）'

        type：控制按钮的类型

          'primary'主要按钮，蓝色

          'danger'危险按钮，红色

          'waring'警告按钮，橙色

          ''默认按钮，灰色

        disable：控制按钮是否被禁用 true/false(默认)

     -->

    <my-button @click="change('hahaha')"> 第一个按钮 </my-button>

    <my-button size="large" type="danger"> 删除 </my-button>

    <my-button type="primary" :disable="true"> 提交 </my-button>

  </div>

</template>

<script>

import MyButton from "./MyButton.vue";

export default {

  components: { MyButton },

  name: "DemoTwo",

  methods: {

    change(type) {

      console.log("点击了:", type);

    },

  },

};

</script>

<style lang="less" scoped>

.demo-box {

  color: pink;

}

:deep(.btn) {

  color: blueviolet;

}

</style>

<template>

  <div class="my-button-box">

    <button

      :class="[size, type, disable ? 'disable' : '']"

      :disabled="disable"

      @click="fire"

    >

      <slot></slot>

    </button>

  </div>

</template>

<script src="">

export default {

  name: "MyButton",

  props: {

    size: {

      type: [String, Number], // 设置传递属性值的类型，如果传递的值不是这个类型，控制台会抛出警告错误，但是不影响正常的获取；如果类型可以是多个中的一种，则设置为数组即可 [String, Number]

      required: true, // 设置为必传

      //   default: "small", // 不穿递属性值，设置的默认值

    },

    type: String, // 只需要控制类型，其余不考虑，

  },

  created() {

    console.log(this.$attrs);

    console.log(this.$listeners);

  },

};

</script>

<script>

export default {

  name: "MyButton",

  props: {

    size: {

      type: String,

      default: "small",

    },

    type: {

      type: String,

      default: "",

    },

    disable: {

      type: Boolean,

      default: false,

    },

  },

  methods: {

    fire() {

      // 通知：之前基于@change给实例事件中加入的自定义事件“名字：click”执行

      this.$emit("click", "hehehe");

    },

  },

};

</script>

<style lang="less" scoped>

.my-button-box {

  position: relative;

  button {

    border: none;

    background: #c0c4cc;

    color: #fff;

    font-size: 14px;

    cursor: pointer;

    &.small {

      padding: 5px 10px;

    }

    &.large {

      padding: 10px 20px;

      font-size: 16px;

    }

    &.primary {

      background: #409eff;

    }

    &.danger {

      background: #f56c6c;

    }

    &.waring {

      background: #e6a23c;

    }

    &.disable {

      background: #606266;

      cursor: initial;

    }

  }

}

</style>